ASSIGNMENT 1

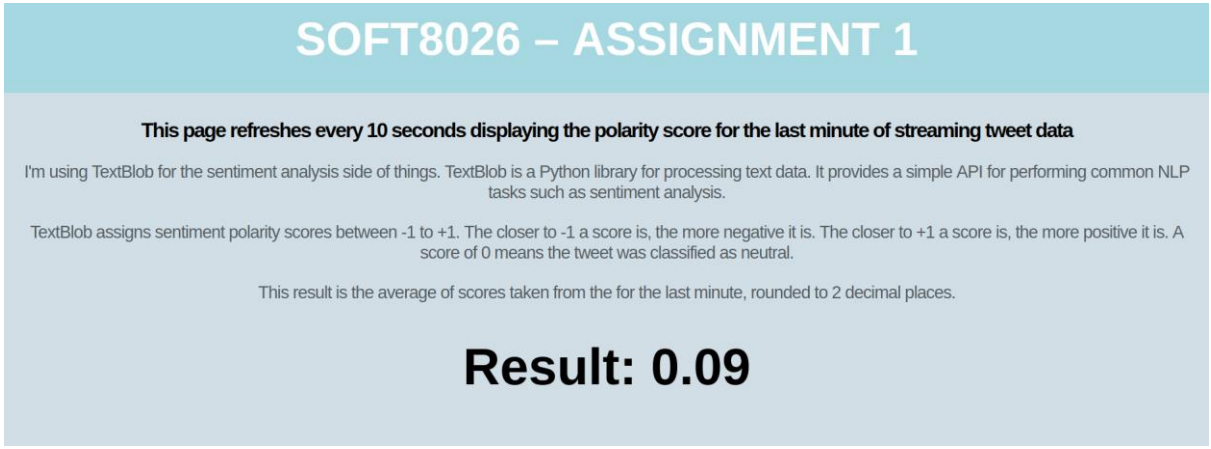
SOFT8026 – Data Driven Microservices

ABSTRACT
Build and deploy a data streaming application which performs basic sentiment analysis. The application must use a microservice architecture.

Peter Burton
R00038147

Provisioning the application

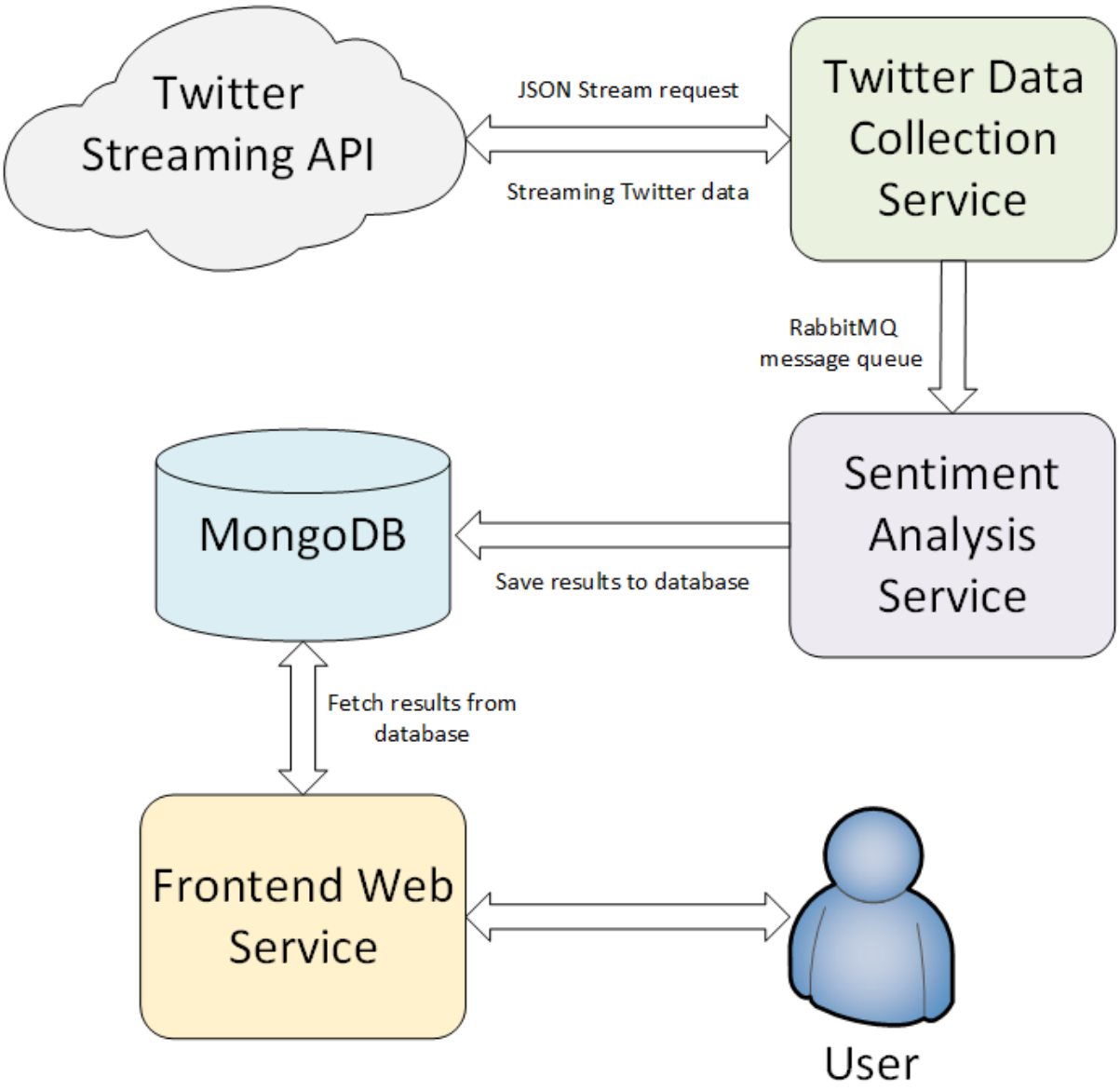
To start the application, navigate to the microservices_assignment1 folder where the docker_compose.yml file is. Open a terminal and run `docker-compose up`. At this point Docker Compose should do its thing and start up 5 containers. The tweet_collection, sentiment_analysis and frontend services that I wrote, and the RabbitMQ and mongoDB services that are vanilla from the Docker Hub. Once the services have finished loading, (which on my laptop took just over 5 minutes), you can access the webpage by entering http://localhost:3000/sentiment in a browser. This should open up the webpage as shown below.



I ran this in my Linux VM and it worked fine. When I tried to run it on Windows it didn't work. From what I can see on the internet, Windows 10 Home edition cannot use the latest version of Docker, and instead has to make do with Docker toolbox. This version uses VirtualBox instead of Hyper-V to implement its VM's and apparently this causes some issues with the network side of things, with services not finding each other and so on. In case of any issues when trying to test out the program I have uploaded a video to YouTube, https://youtu.be/pDIy1vPSCQg, which shows the output from running the `docker-compose up` command on my VM and shows the web application up and running and the result value changing roughly every 10 seconds.

System Architecture

System Diagram



System Overview

I created three microservices for this assignment. A service to collect tweets and publish them to RabbitMQ, a second service to take the tweets from RabbitMQ, analyse their sentiment and save the result to MongoDB, and a third service to display the results from MongoDB in a web application. The architecture of this system is quite clean and easy to containerize, with clear boundaries between each service. The very high-level logic of each service could be described like this:

1. Get some information.
2. Do something with it.
3. Pass on the updated information.

It would be easy enough to swap out any of the services for an implementation in a different language and it would still work as the logic required to run any one service doesn't depend on any of the other services, and the communication mechanism used (RabbitMQ) is universally compatible with many different languages. I feel that these three services were an ideal number for this simple application. If the application was more complicated/had more requirements, I might think about implementing a backend REST API service that would fit in between the sentiment service and the database, and also between the front end and the database, but it seemed pointless in this case as the sentiment analysis service just has to dump its results into the database as it analyses the sentiment scores, and the frontend service is also not doing too much processing before rendering the webpage.

The data collection service is written in Go. The collect_publish.go program connects to Twitter via the streaming API and requests a JSON stream about a search term, (in this case the search term is "trump" and is hardcoded). On receiving the stream, it extracts the body of the tweet, and publishes it to the RabbitMQ message queue service. RabbitMQ is a message broker type service where a "producer" can publish messages to the message queue, and the "consumer" can subscribe to that particular queue, and consume the messages from it, as shown below taken from RabbitMQ's website:

The sentiment analysis service is written in Python. This service subscribes to the RabbitMQ message queue, and reads the tweets from it using the pika library. Pika is a pure Python implementation of the AMQP 0-9-1 protocol including RabbitMQ's extensions. The sentiment.py program subscribes to the RabbitMQ message queue via pika using a simple blocking connection. It reads in the stream of tweets and uses the TextBlob library for its sentiment analysis. TextBlob is a Python library for processing textual data. It provides a simple API for common natural language

processing (NLP) tasks such as sentiment analysis and classification. TextBlob assigns sentiment polarity scores between -1 to +1. The closer to -1 a score is the more negative the sentiment. The closer to +1 a score is the more positive the sentiment. A score of 0 indicates that the tweet was classified as neutral. The program then saves the score for the tweet along with a timestamp to the database.

The frontend service is also written in Python and uses the Flask web framework. It runs a webpage that refreshes every 10 seconds. The front_end.py program pulls a list from the database of the last minutes results according to their timestamps. It then performs a calculation to get the average of the results, formats it, and renders the webpage with the new result. In practice, looking at the output from the terminal the webpage seemed to update more like every 15 seconds, but I'm assuming that is where the laptop was struggling to run everything.

Wait for it!

I ran into a problem where even though I had specified that the twitter_collection service and the sentiment_analysis service depended on the RabbitMQ service, Docker was starting them before RabbitMQ was ready to receive connections. This is because Docker doesn't care if the program within the container is ready to accept connections, it just assumes that once the container has successfully started it's OK to start any of its dependant containers. To get around this issue I used the wait-for-it.sh script which is available here: https://github.com/vishnubob/wait-for-it. This is a bash script that tests the availability of a TCP host and port. This made it easy to control the start-up order as I could specify in the docker-compose.yml file a command that would run the wait-for-it script until RabbitMQ was ready and then start-up the dependant containers. This is the command:

```
command: ["./wait-for-it.sh", "-t", "0", "rabbitmq:5672", "--", "go", "run",
"collect_publish.go"]
```